Norges teknisk-naturvitenskapelige universitet

Institutt for datateknikk og informasjonsvitenskap

Løsningsforslag til

EKSAMENSOPPGAVE I FAG TDT4186 – OPERATIVSYSTEMER

Versjon: 20. nov 2012

Faglig kontakt under eksamen: Svein Erik Bratsberg og Arvid Staupe

Tlf.: 99539963 og 73591725

Eksamensdato: **9. desember 2010**

Eksamenstid: **09.00-13.00**

Tillatte hjelpemiddel: **D: Ingen trykte eller håndskrevne hjelpemiddel tillatt. Bestemt, enkel kalkulator tillatt.**

Språkform: Bokmål

Sensurdato: 10. januar 2011

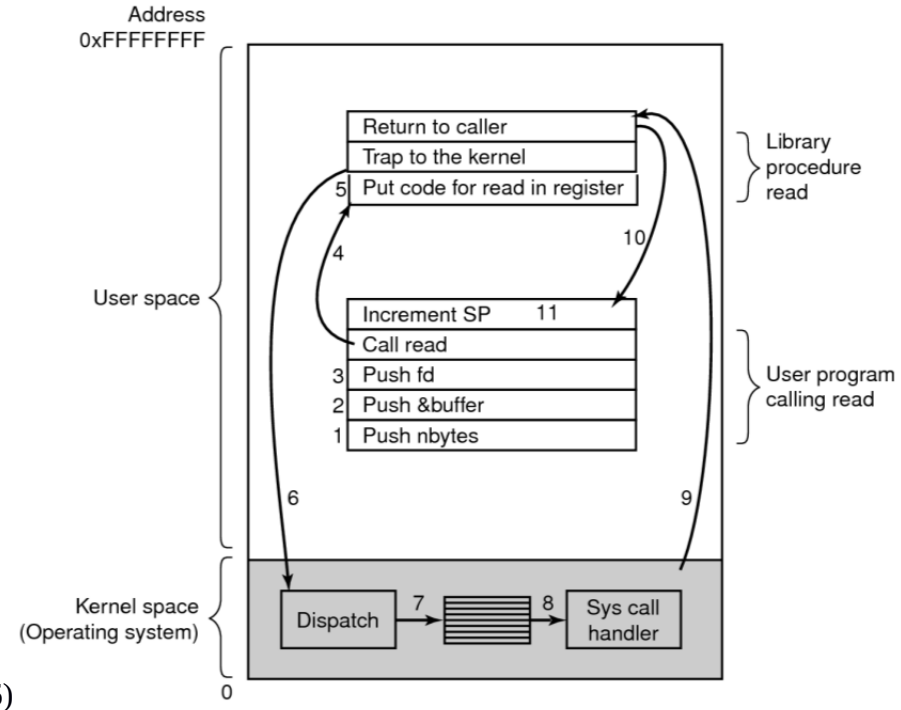
Oppgave 1 – Generelt, prosesser og sikkerhet – 15 %

a) (Kap 1.1) Et operativsystem er en utvidet datamaskin som gjemmer detaljer og er lett å bruke. Det er også en ressurshåndterer som tar seg av tids- og plassdeling.

b) (Kap 2.1.6?) Vi må lagre unna registere ved prosesskifte fordi prosessen kan være midt i beregninger når den blir avbrutt. F.eks. kan tilordning av to 128bits heltallsvariable (a = b;) være delt opp i fire 64 bits-instruksjoner (load, load, store, store). Et prosesskifte kan skje før, etter eller mellom instruksjonene. Registere brukes for eksempel til mellomlagringer for beregninger og har derfor ingen tilhørende plass i datadelen av adresserommet.

c) (Kap 9.7) «Malware» omfatter trojanske hester, virus, ormer, spyware og root kits. Pluss hvis studenten forklarer hva hver av disse er.

Oppgave 2 – Systemkall – 10 %

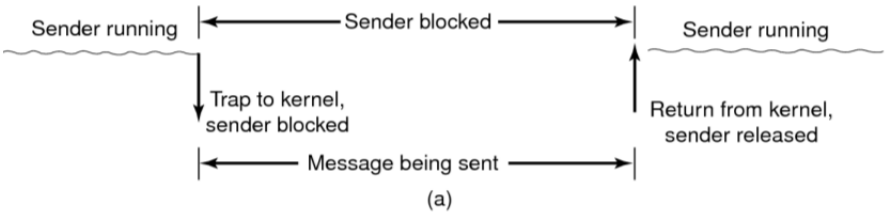


a) (Kap 1.6) Vi skifter til kjernemodus for å kunne gjøre ting som krever privilegier, som for eksempel I/O. Forklaring av figuren er bra.

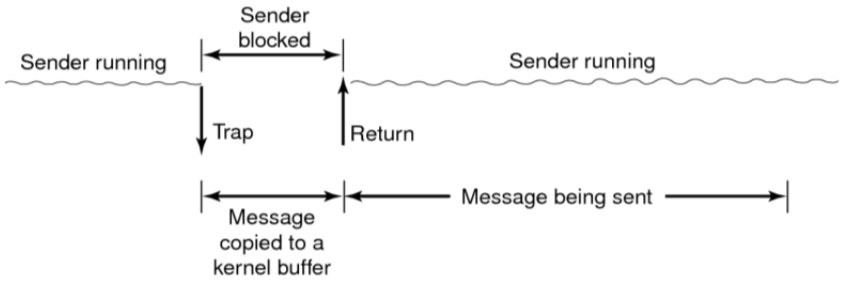
b) (Kap 1.6) Vi har systemkall for prosesser som f.eks. fork, exit, wait og execv. Vi har systemkall for I/O som read, seek, write, open, close, recv og send. Vi har systemkall for kataloghåndtering som mkdir, rmdir, link og unlink.

Oppgave 3 – Meldingssending – 10 %

a) (Kap 8.2.3) send kan være både synkron og asynkron. Synkron send blokkerer senderen slik at den først får fortsette når meldingen er sendt. Asynkron send returnerer omtrent umiddelbart, evt. etter at meldingen er kopiert til et kjernebuffer. Tilsvarende for recv betyr synkron at prosessen/kjernetråden venter på en innkommende melding før den returnerer, mens asynkron betyr at den kan returnere før noen melding er kommet.



(a)



(b)

b) (Kap 8.2.3) Fordelen med asynkron håndtering er at senderen/mottakeren kan gjøre andre oppgaver så lenge, slik at throughput'en kan være større. Ulempen er at det blir vanskeligere å håndtere meldingsbuffere, da du ikke kan vite om meldingen har kommet av gårde før du gjenbruker bufferne.

Oppgave 4 – Minnehåndtering – 20 %

a) (Kap 3.4.2) Under er første rad referansene. Rad 2 er første ramme, rad 3 er andre ramme, osv. r og m-bitene er vist rett etter nummeret på den virtuelle siden. NRU gir 8 sidefeil her (vist i rødt). Det er viktig at studentene skjønner at modifiserte sider ikke kan stjeles.

0(r)	1(w)	2(r)	Cl	4(r)	5(r)	6(w)	Cl	0(r)	1(w)	5(r)	Cl	6(w)	0(r)
0r	0r	0r	0	0	5r	5r	5	0r	0r	0r	0	0	0r
	1rm	1rm	1m	1m	1m	1m	1m	1m	1rm	1rm	1m	1m	1m
		2r	2	2	2	6rm	6m	6m	6m	6m	6m	6tm	6rm
				4r	4r	4r	4	4	4	5r	5	5	5

b) (Kap 3.4.6) LRU. Her har vi vist rammene som en stakk, men i praksis vil vi ikke kopiere mellom rammene i minnet, men kun vedlikeholde en referansestruktur. Her har vi antatt at modifiserte sider ikke kan stjeles siden ingen av de modifiserte rammene skrives ut i denne perioden. Da får vi totalt 7 sidefeil her.

0(r)	1(w)	2(r)	Cl	4(r)	5(r)	6(w)	Cl	0(r)	1(w)	5(r)	Cl	6(w)	0(r)
				0	1m	1m	1m	1m	5	6m	6m	0	1m
		0	0	1m	2	4	4	5	6m	0	0	1m	5
	0	1m	1m	2	4	5	5	6m	0	1m	1m	5	6rm
0	1m	2	2	4	5	6m	6m	0	1m	5	5	6m	0

c) (Kap 3.4.9) WSClock gir 7 sidefeil. *I retrospekt er denne oppgaven detaljert i forhold til den løse beskrivelsen i boka.* Under har vi illustrert dette med 14 tabeller. Rad 1 er referansen. Kolonne 1 er rammene med R- og M-bitene. Kolonne 2 er referansetidspunktet. Kolonne 3 er viseren etter at sidefeilen er utført.

Vi har her brukt følgende regler for når «Time of last use» oppdateres.

1. Hvis R-bitet er satt når klokkeavbruddet går, settes referansetidspunktet til current virtual time.
2. Når sidefeilalgoritmen (WSClock) sjekker om et element skal kastets ut, og R-bitet for elementet er satt, settes referansetidspunktet for elementet til current virtual time.
3. Når ei side feiles inn, settes referansetidspunktet til current virtual time.

Slik det ender ut her, blir ikke tau brukt da alle ledige sider har referansetidspunkt nyere enn tau.

0(r)	**0**	
0r	0	
		--

1(w)	**30**	
0r	0	
1rm	30	
		--

2(r)	**60**	
0r	0	
1rm	30	
2r	60	

		--

Cl	**90**	
0	90	
1m	90	
2	90	
		--

4(r)	**120**	
0	90	--
1m	90	
2	90	
4r	120	

5(r)	**150**	
5r	150	
1m	90	--
2	90	
4r	120	

6(w)	**180**	
5r	150	
1m	90	
6rm	180	
4r	120	--

Cl	**210**	
5	210	
1m	90	
6m	210	
4	210	--

0(r)	**240**	
5	210	--
1m	90	
6m	210	
0r	240	

1(w)	**270**	
5	210	--
1rm	90	
6m	210	
0r	240	

5(r)	**300**	
5r	210	--
1rm	90	
6m	210	
0r	240	

Cl	**330**	
5	330	--
1m	330	
6m	210	
0	330	

6(w)	**360**	
5	330	--
1m	330	
6rm	210	
0	330	

0(r)	**390**	
5	330	--
1m	330	
6rm	210	
0r	330	

Oppgave 5 – Filsystemer – 10%

a) (Kap 4.3.2) FAT-filsystem kan ikke ha ”hull”, så derfor allokeres det 1 000 004/4096 = 245 blokker:

b) (Kap 4.5.3) 4 havner i blokk 0, 1004 havner i blokk 0. 100 004 havner i blokk 24, som da krever en indirekte blokk også. 1 000 004 havner i blokk 244. Denne vil være pekt på av samme indirekteblokk som den forrige, da det er plass til 1024 pekere i enkeltindirekteblokka. Derfor får vi allokert 4 blokker på disken, i tillegg til inoden.

Oppgave 6 – Vranglås – 15%

a) (6.5.2) En usikker tilstand betyr ikke at vi har vranglås, men det betyr at vi kan få vranglås hvis alle prosesser spør om sitt maksimale behov for ressursene samtidig. En usikker tilstand kan muligens gi en vranglås.

b) (6.4.3) Oppgaven antar at det er et **delt** buffer mellom to prosesser. Den ene prosessen må gi fra seg en bufferplass slik at den andre kan komme videre. Dette kan enten skje ved sjekkpunkt/restart eller ved å drepe prosessen.

c) (6.6.2) Hvis alle prosesser spør om sitt maksimale behov før de starter, vil det ikke oppstå vranglås. Ulempene med dette er at det er vanskelig å forutsi det maksimale behov på forhånd, og det gir dårlig ressursutnyttelse. Noen studenter har også forslått å nummere bufferne og ta de i samme rekkefølge.

Oppgave 7 – Multiprosessorsynkronisering – 10%

a) (Kap 8.1.3)

1. Det er ikke mulig å bruke ”disable interrupts” som løsning for synkronisering i multiprosessorsystemer med delt minne. Dette skrur av avbrudd kun for en CPU. I stedet må det brukes Test-and-Set-Lock (eller tilsvarende synkroniseringsinstruksjoner).

2. De andre CPU-ene vil spinne når den venter på låsen, noe som forårsaker bortkastet CPU-tid.

3. Det er problem at CPU-caching forårsaker mye busstrafikk mens CPU-ene står og spinner på en lås. Problemet kommer av at den som har låsen endrer på data i nærheten av låsen mens de andre CPU-ene står og spinner og må hente de endrede cachelinjene på nytt hver gang de endres.

b) (Kap 8.1.3) Hvordan løses problemene?

1. Vi må bruke TSL (Test-and-Set-Lock). Det er viktig at både lesingen og skrivingen foregår som en atomisk instruksjon. Derfor må bussen låses når dette skjer.

2. Spinning kan være greit når det foregår over kort tid, men etter hvert bør CPU-en skifte til en annen tråd/prosess som har fornuftig arbeid å gjøre.

3. Her kan det lages en egen kopi av låsen for hver CPU som spinner. Disse kopiene kjedes sammen slik at kun en og en vekkes opp når låsen slippes.

Oppgave 8 – I/O – 10%

a) (Kap 5.1.3) Fordeler: 1. kan bruke høynivåspråk for devicedrivere. 2. Enklere å ordne beskyttelse. 3. Mange av instruksjonene kan bruke minne. Ulemper: 1. Må skru av caching for deler av minnet. 2. Alle I/O-enheter må ”snuse” på alle minnereferanser. Spesiell hardware for dette.

b) (Kap 2 og 5) Spørsmåler er bittelitt vagt da det egentlig spørres om “per sekund”. Men det burde være klart for de som har gjort den nesten identiske øvingsoppgaven. Tiden det tar å pushe og poppe alle registrene på/fra stakken er alene (20+2)*10*2 nanosek = 440 nanosek. Dette alene betyr at vi ikke kan greie mer enn 2,27 millioner avbrudd per sekund.